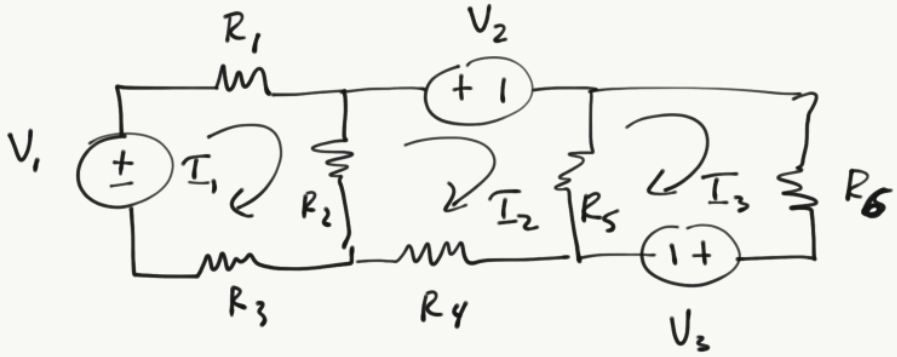
Linear Algebra

- Sets of linear equations play a key role in many physical systems — these can be represented as matrices
- For example analysis of circuits:



(I_i unknown)

Applying Kirchoff's voltage laws around each loop:

$$V_1 - I_1 R_1 - (I_1 - I_2) R_2 - I_1 R_3 = 0$$

$$-V_2 - (I_2 - I_3) R_5 - I_2 R_4 - (I_2 - I_1) R_2 = 0$$

$$-(I_3 - I_2) R_5 - I_3 R_6 - V_3 = 0$$

Rearranging, we find:

$$(-R_1 - R_2 - R_3) I_1 + R_2 I_2 + 0 = -V_1$$

$$+R_2 I_1 + (-R_5 - R_4 - R_2) I_2 + R_5 I_3 = V_2$$

$$+0 + R_5 I_2 + (-R_5 - R_6) I_3 = V_3$$

- More generally a system of 3 eqns. & 3 unknowns:

$$A_{11}x_1 + A_{12}x_2 + A_{13}x_3 = b_1$$

$$A_{21}x_1 + A_{22}x_2 + A_{23}x_3 = b_2$$

$$A_{31}x_1 + A_{32}x_2 + A_{33}x_3 = b_3$$

- These can be represented in matrix form as:

$$\underline{\underline{A}}\,\underline{x} = \underline{b}$$

(note compactness!)

A_{ij} are elements of $\underline{\underline{A}}$
x_j are elements of $\underline{x}$
b_i are elements of $\underline{b}$

- Referring back to the system above & enforcing equivalence with matrix form, it is clear that matrix multiplication is defined by:

$$\left(\underline{\underline{A}}\,\underline{x}\right)_i \equiv \sum_j A_{ij}x_j$$

→ more generally for two matrices

$$\boxed{\left(\underline{\underline{A}}\,\underline{\underline{B}}\right)_{ij} \equiv \sum_k A_{ik}B_{kj}}$$

matrix multiplication

- For multiplication to make sense so matrices must be conformable, viz. of size: $(m \times p)\cdot(p \times n)$

inner dimensions equal!

$m \times n$

- Mathematically speaking, a system of equations could fall into one of four categories:

  1) no solution (inconsistent)
  2) trivial solution $\underline{x} = \underline{0}$
  3) infinite solutions conforming to some constraint
  4) unique solution

- For physical situations we expect 4) !!!
- Linear systems play a large role in numerical analysis - networks, fitting & estimation of parameters, interpolation, & ODEs/PDEs

- Special Matrices:

  1) column vector $\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$

  2) row vector $\underline{y} = [y_1 \; y_2 \; \dots \;]$

  by notational default we assume $\underline{x}$ is column & $\underline{x}^T$ is row, unless otherwise stated...

  3) square matrix has size $n \times n$

  4) diagonal matrix $\underline{\underline{D}} = \begin{bmatrix} d_{11} & & 0 \\ & d_{22} & \\ 0 & & \ddots \end{bmatrix}$

identity matrix $\underline{\underline{I}} = \begin{bmatrix} 1 & & 0 \\ & 1 & \\ 0 & & \ddots \\ & & & 1 \end{bmatrix}$

upper triangular $\underline{\underline{U}} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots \\ & u_{22} & u_{23} & \\ 0 & & \ddots & \end{bmatrix}$

lower triangular $\underline{\underline{L}} = \begin{bmatrix} l_{11} & & 0 \\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33} \cdots \end{bmatrix}$

banded $\underline{\underline{B}} = \begin{bmatrix} b_{11} & 0 & b_{13} & 0 \\ 0 & b_{22} & 0 & b_{24} \\ b_{31} & 0 & b_{33} & 0 \\ 0 & b_{42} & 0 & b_{44} \end{bmatrix}$

- Matrix operations all follow from linear equation properties

$$(\underline{\underline{A}} + \underline{\underline{B}})_{ij} = A_{ij} + B_{ij} \; ; \; (\underline{\underline{A}} - \underline{\underline{B}})_{ij} = A_{ij} - B_{ij}$$

$\underline{\underline{A}}, \underline{\underline{B}}$ must have same size!

$$(\alpha \underline{\underline{A}})_{ij} = \alpha A_{ij}$$

$$\underline{\underline{A}}(\underline{\underline{B}} + \underline{\underline{C}}) = \underline{\underline{A}}\underline{\underline{B}} + \underline{\underline{A}}\underline{\underline{C}} \; ; \; \boxed{\underline{\underline{A}}\underline{\underline{B}} \neq \underline{\underline{B}}\underline{\underline{A}}}$$

- Matrix inverse $\underline{\underline{A}}^{-1}$ defined s.t :

$$\underline{\underline{A}}\,\underline{\underline{A}}^{-1} = \underline{\underline{A}}^{-1}\,\underline{\underline{A}} = \underline{\underline{I}}$$

- Solutions to $\underline{\underline{A}}\,\underline{\underline{B}} = \underline{\underline{C}}$ for $\underline{\underline{A}}$:

$$\underline{\underline{A}} = \underline{\underline{B}}^{-1}\underline{\underline{C}}$$

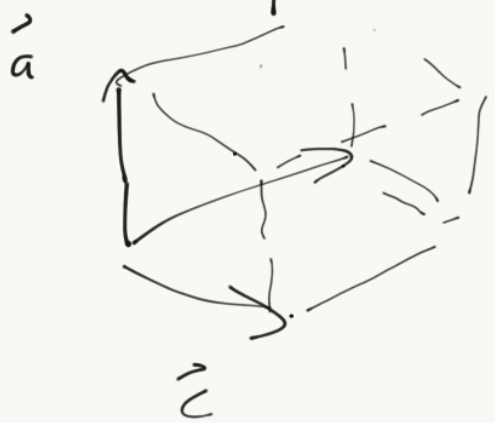
similarly $\underline{\underline{A}}\,\underline{x} = \underline{b} \rightarrow$ $\boxed{\underline{x} = \underline{\underline{A}}^{-1}\,\underline{b}}$

- Determinants of matrices can be thought of as analogous to concept of "volume" or magnitude

$$\underline{\underline{A}} = \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix} \qquad \vec{a} = (a_1, a_2, a_3)$$

$\vec{b} = \ldots$ etc.

$\det(\underline{\underline{A}}) \equiv |\underline{\underline{A}}|$ is related to the triple product $\vec{a}\cdot(\vec{b}\times\vec{c})$ which is volume spanned by $\vec{a}, \vec{b}, \vec{c}$



if $|\underline{\underline{A}}| = 0$

then $\vec{a}$ lies in plane of $\vec{b}, \vec{c}$

linearly dependent

- More generally we can define $|\underline{\underline{A}}|$ by:

$$|\underline{\underline{A}}| = \sum_{j=1}^{n} A_{ij}\, \text{cof}(A_{ij}) = \sum_{j=1}^{n} A_{ij}\,(-1)^{(i+j)}\, M_{ij}$$

(i arb) ↙ $j=1$ ↓ cofactor of A_{ij} $j=1$ ↓ minor of A_{ij}

M_{ij} is the minor defined by evaluation det of augmented matrix, without row i, column j

- Note that det(scalar) = scalar.
- Consider a 2×2 determinant

$$|\underline{\underline{A}}| = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} = A_{11}(-1)^{1+1} M_{11} + A_{12}(-1)^{1+2} M_{12}$$

$$M_{11} \rightsquigarrow \begin{vmatrix} \cancel{A_{11}} & \cancel{A_{12}} \\ \cancel{A_{21}} & A_{22} \end{vmatrix} = \det(A_{22}) = A_{22}$$

$$M_{12} \rightsquigarrow \begin{vmatrix} \cancel{A_{11}} & \cancel{A_{12}} \\ A_{21} & \cancel{A_{22}} \end{vmatrix} = \det(A_{21}) = A_{21}$$

$$\boxed{|\underline{\underline{A}}| = A_{11}A_{22} - A_{12}A_{21}} \quad (2\times 2 \text{ det})$$

- Can also expand cofactors along column:

$$\boxed{|\underline{\underline{A}}| = \sum_{i=1}^{n} A_{ij}\, \text{cof}(A_{ij})} \quad (j \text{ fixed})$$

- 3×3 determinants proceed similarly:

$$|\underline{\underline{A}}| = A_{11}(-1)^{1+1}M_{11} + A_{12}(-1)^{1+2}M_{12} + A_{13}(-1)^{1+3}M_{13}$$

$$M_{11} \sim \begin{vmatrix} \cancel{A_{11}} & \cancel{A_{12}} & \cancel{A_{13}} \\ A_{21} & A_{22} & A_{23} \\ \cancel{A_{31}} & A_{32} & A_{33} \end{vmatrix} = A_{22}A_{33} - A_{23}A_{32}$$

$$M_{12} = A_{21}A_{33} - A_{23}A_{31} \;;\; M_{13} = A_{21}A_{32} - A_{22}A_{31}$$

$$|\underline{\underline{A}}| = A_{11}(A_{22}A_{33} - A_{23}A_{32}) - A_{22}(A_{21}A_{33} - A_{23}A_{31}) + A_{13}(A_{21}A_{32} - A_{22}A_{31})$$

- Evaluating an $n\times n$ determinant requires some adds & $n!$ terms, each containing $(n-1)$ multiplications, so it scales like: adds

$$\mathcal{O}((n-1)n!)$$ (cofactor expansion of det)

Elimination Methods

- Cramer's rule is a direct method & useful for establishing a baseline approach for comparing efficiency of various methods:

$$\underline{\underline{A}}\,\underline{x} = \underline{b} \quad ; \quad x_j = \frac{|\underline{\underline{A}}^j|}{|\underline{\underline{A}}|} \quad ; \quad (\underline{\underline{A}}^j \text{ is } \underline{\underline{A}} \text{ with the } j^{th} \text{ column replaced by } \underline{b})$$

- e.g. $x_1 = \dfrac{\begin{vmatrix} b_1 & A_{12} \\ b_2 & A_{22} \end{vmatrix}}{A_{11}A_{22} - A_{12}A_{21}}$; $\dfrac{\begin{vmatrix} A_{11} & b_1 \\ A_{21} & b_2 \end{vmatrix}}{A_{11}A_{22} - A_{12}A_{21}} = x_2$

$$x_1 = \frac{b_1 A_{22} - A_{12} b_2}{A_{11}A_{22} - A_{12}A_{21}} \; ; \quad x_2 = \frac{A_{11} b_2 - b_1 A_{21}}{A_{11}A_{22} - A_{12}A_{21}}$$

- Dealing with larger systems is problematic. For $\underline{\underline{A}} \sim n \times n$ we must evaluate $n+1$ determinants.
- Each determinant has $n!$ products (all containing n items multiplied) So $(n!)(n-1)$ multiplications per det.
- Total multiplications:

$$|\underline{\underline{M}}| = \mathcal{O}((n+1)(n-1)\,n!)$$

$$= \mathcal{O}((n-1)(n+1)!)$$

- Thus, Cramer's rule is too computationally expensive to practically use. (SHOW IN MATLAB).
- Elimination methods address this shortcoming; these operate very much like a normal solution method by hand.
  1) perform algebraic substitutions until value of one variable can be found (fwd elimination)
  2) substitute this variable back into simplified eqns successively to find other vars. (back substitution)

- Elementary row operations (allow elimination steps)
  1) Any row can be multiplied by scalar (scaling)
  2) Rows can be interchanged (pivoting)
  3) Rows may be linearly combined with other rows. (elimination)
- None of these rows operations change the solution to the system $\underline{\underline{A}}\,\underline{x} = \underline{b}$

- Connection of row operations to matrix multiplication, e.g.:

1) $\underline{\underline{E}}_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$; $\underline{\underline{E}}_1 \underline{\underline{A}} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ \alpha A_{21} & \alpha A_{22} & \alpha A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$

2) $\underline{\underline{E}}_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$; $\underline{\underline{E}}_2 \underline{\underline{A}} = \begin{bmatrix} A_{21} & A_{22} & A_{23} \\ A_{11} & A_{12} & A_{13} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$

$$\underline{\underline{E}}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \alpha & 0 & 1 \end{bmatrix}; \quad \underline{\underline{E}}_3 \underline{\underline{A}} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ \alpha A_{11} + A_{31} & \alpha A_{12} + A_{32} & \alpha A_{13} + A_{33} \end{bmatrix}$$

- Because of the connection of row ops to multiplication we can conceive as an inverse as a sequence of row ops.

$$\left.\begin{aligned} \underline{\underline{A}}^{-1} \underline{\underline{A}} &= \underline{\underline{I}} \\ \left(\prod_i \underline{\underline{E}}_i\right) \underline{\underline{A}} &= \underline{\underline{I}} \end{aligned}\right\}$$

so its possible to find an inverse hence solution from row operations!!!

- So if we find row ops that reduce $\underline{\underline{A}}$ to $\underline{\underline{I}}$ we can apply those ops to $\underline{\underline{I}}$ to get $\underline{\underline{A}}^{-1}$:

$$\underline{\underline{A}}^{-1} = \left(\prod_i \underline{\underline{E}}_i\right) \cdot \left(\prod_i \underline{\underline{E}}_i\right) \underline{\underline{I}}$$

NOTE: Matrix multiplication is associative!

$$\underline{\underline{A}}\left(\underline{\underline{B}}\underline{\underline{C}}\right) = \left(\underline{\underline{A}}\underline{\underline{B}}\right)\underline{\underline{C}}$$

$$\begin{aligned} x_1 + 4x_2 + 2x_3 &= 15 \\ 3x_1 + 2x_2 + x_3 &= 10 \\ 2x_1 + x_2 + 3x_3 &= 13 \end{aligned} \rightarrow \left[\begin{array}{ccc|c} 1 & 4 & 2 & 15 \\ 3 & 2 & 1 & 10 \\ 2 & 1 & 3 & 13 \end{array}\right]$$

elim. from row 1:

$$\left[\begin{array}{ccc|c} 1 & 4 & 2 & 15 \\ 3 & 2 & 1 & 10 \\ 2 & 1 & 3 & 13 \end{array}\right] \quad \begin{array}{l} R_2 - 3R_1 \\ R_3 - 2R_1 \end{array}$$

elim. from row R_2:

$$\left[\begin{array}{ccc|c} 1 & 4 & 2 & 15 \\ 0 & -10 & -5 & -35 \\ 0 & -7 & -1 & -17 \end{array}\right] \quad R_3 - \tfrac{7}{10} R_2$$

$$\begin{aligned} -1 - \tfrac{7}{10}(-5) &= 5/2 \\ -17 - \tfrac{7}{10}(-35) &= -17 + \tfrac{245}{10} \\ &= \tfrac{-170+245}{10} \\ &= \tfrac{75}{10} = \tfrac{15}{2} \end{aligned}$$

final state following elim:

$$\left[\begin{array}{ccc|c} 1 & 4 & 2 & 15 \\ 0 & -10 & -5 & -35 \\ 0 & 0 & 5/2 & 15/2 \end{array}\right]$$

backsubstitution:

$$x_3 = (15/2)\left(\tfrac{2}{5}\right) = 3$$

$$x_2 = (-35 + 5(3))\left(\tfrac{-1}{10}\right) = -20\left(\tfrac{-1}{10}\right) = 2$$

$$x_1 = (15 - 2(3) - 4(2)) = 1$$

- Two modifications to standard elimination schemes are advisable: pivoting & scaling.

- The pivot element for an elimination step is the element being used for elimination. e.g.

pivot

$$\left[\begin{array}{ccc|c} \textcircled{3} & 2 & 1 & 10 \\ 2 & 1 & 3 & 13 \\ 1 & 4 & 2 & 15 \end{array}\right] \quad \begin{array}{l} R_2 - \left(\frac{2}{3}\right) R_3 \\ R_3 - \left(\frac{1}{3}\right) R_3 \end{array}$$

elimination multiplier

- Interchanging rows (reordering eqns.) doesn't change solution — so one can switch pivots with impunity

- For reasons of numerical accuracy, it is always best to scale each row by largest element in that row & then switch to largest pivot.

$$\left[\begin{array}{ccc|c} 1 & 4 & 2 & 15 \\ 3 & 2 & 1 & 10 \\ 2 & 1 & 3 & 13 \end{array}\right] \quad \begin{array}{l} \text{pivot: } \frac{1}{15} \\ \text{pivot: } \frac{3}{10} \\ \text{pivot: } \frac{2}{13} \end{array} \quad \text{swap}$$

$$\downarrow$$

$$\left[\begin{array}{ccc|c} 3 & 2 & 1 & 10 \\ 1 & 4 & 2 & 15 \\ 2 & 1 & 3 & 13 \end{array}\right]$$

do elimination, then reorder again for largest scaled pivot

defines Gaussian elimination

- Gauss – Jordan elimination is based on the fact that one may compute inverse as a sequence of row ops on $\underline{\underline{I}}$:

$$\underline{\underline{A}}^{-1}\underline{\underline{A}} = \underline{\underline{I}} \rightarrow \left(\prod_i \underline{\underline{E}}_i\right)\underline{\underline{A}} = \underline{\underline{I}}$$

$$\therefore \left(\prod_i \underline{\underline{E}}_i\right)\underline{\underline{I}} = \underline{\underline{A}}^{-1}$$

$$\left[\begin{array}{ccc|ccc} 1 & 4 & 2 & 1 & 0 & 0 \\ 3 & 2 & 1 & 0 & 1 & 0 \\ 2 & 1 & 3 & 0 & 0 & 1 \end{array}\right] = \left[\underline{\underline{A}} \,\middle|\, \underline{\underline{I}}\right] \quad \begin{array}{l} R_2 - 3R_1 \\ R_3 - 2R_1 \end{array}$$

- See book examples & homework.
- Generally elimination methods of all types are more efficient than solving a system via Cramer's Rule; which scaled like

$$\mathcal{O}((n-1)(n+1)!) \qquad \underline{\text{Cramer's Rule}}$$

- E.g. Gauss elimination scales like: $\mathcal{O}\left(\frac{n^3}{3} - \frac{n}{3}\right)$

Determinants

- Determinants can be evaluated efficiently using elimination methods.
- Consider an upper triangular matrix resulting from elimination:

$$\underline{\underline{U}} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots \\ 0 & u_{22} & u_{23} & \cdots \\ 0 & 0 & u_{33} & \\ 0 & 0 & 0 & \ddots \end{bmatrix}$$

$$\det(\underline{\underline{U}}) = (\text{expand along } 1^{\text{st}} \text{ column})$$

$$= u_{11}(-1)^{1+1}\left|\underline{\underline{M_{11}}}\right| = u_{11}(-1)^{1+1}\left[u_{22}(-1)^{1+1}\left|\underline{\underline{M_{22}}}\right|\right]$$

(expand along 1^{st} column of $\underline{\underline{M_{11}}}$)

$$= \ldots = \prod_i u_{ii}$$

- So evaluating dets of triangular matrices is trivial
- Performing simple elimination does not change the determinant, thus we can do fwd elimination then evaluate det from result.
- This is more efficient than the $(n-1)n!$ operations implied by cofactor expansions.
- Pivoting changes value of determinants in predictable ways:
  1) Scaling row by a const also scales det by const
  2) Swapping rows flips sign of det.

LU factorization

- One can factor a matrix in infinite ways; one useful way is to factor into upper & lower triangular matrices:

$$\underline{\underline{A}} = \underline{\underline{L}}\,\underline{\underline{U}}$$

- This can be used to solve $\underline{\underline{A}}\,\underline{x} = \underline{b}$ as follows:

$$\underline{\underline{L}}\,\underline{\underline{U}}\,\underline{x} = \underline{b}\;; \qquad \underline{\underline{L}}^{-1}\underline{\underline{L}}\,\underline{\underline{U}}\,\underline{x} = \underline{\underline{L}}^{-1}\underline{b}$$

$$\underline{\underline{U}}\,\underline{x} = \underline{\underline{L}}^{-1}\underline{b}$$

- Defining $\underline{b}' = \underline{\underline{L}}^{-1}\underline{b}$ we have the following eqns:

(1) $\underline{\underline{L}}\,\underline{b}' = \underline{b}$ (2) $\underline{\underline{U}}\,\underline{x} = \underline{b}'$

- Thus we can view solving $\underline{\underline{A}}\,\underline{x} = \underline{b}$ as solving (1), (2) above
- This is advantageous since triangular systems can be solved easily using fwd/back substitution. Also for systems with multiple RHS, $\underline{\underline{L}}, \underline{\underline{U}}$ need only be computed once!
- An example with multiple RHS would be finding an inverse via LU factorization:

find $\underline{\underline{A}}^{-1}$ s.t. $\underline{\underline{A}}\,(\underline{\underline{A}}^{-1}) = \underline{\underline{I}}$ can be viewed as

$$\left.\begin{aligned} \underline{\underline{A}}\,\underline{x}_1 &= \underline{b}_1 \\ \underline{\underline{A}}\,\underline{x}_2 &= \underline{b}_2 \\ \underline{\underline{A}}\,\underline{x}_3 &= \underline{b}_3 \end{aligned}\right\} \quad \underline{\underline{A}}^{-1} = [\underline{x}_1\ \underline{x}_2\ \underline{x}_3]$$

$\underline{b}_1 = I_{i1}$ (first column of $\underline{\underline{I}}$)

$\underline{b}_2 = I_{i2}$ (2nd " " ")

$\underline{b}_3 = I_{i3}$ (3rd " " ")

Tridiagonal Systems

- Consider the system:

$$\underline{\underline{T}}\,\underline{x} = \underline{b};$$

$$\underline{\underline{T}} = \begin{bmatrix} T_{11} & T_{12} & & & & & \\ T_{21} & T_{22} & T_{23} & & & & \\ & T_{23} & T_{33} & T_{34} & & & \\ & & T_{34} & T_{44} & T_{45} & & \\ & & & \ddots & \ddots & \ddots & \\ & & & T_{i,i-1} & T_{ii} & T_{i,i+1} & \\ & & & & \ddots & \ddots & \\ & & & & & T_{n,n-1} & T_{nn} \end{bmatrix}$$

- Elimination applied to this system is particularly simple. E.g. only one element in row 2 (column 1) to be eliminated using row 1.

$$R_2 \to R_2 - \frac{T_{21}}{T_{11}} R_1; \text{ viz. } T_{2j} = T_{2j} - \frac{T_{21}}{T_{11}} T_{1j}$$

$$j \in \{2,3\}$$

- Now fwd elimination can proceed downward to subsequent rows...

- We must also alter RHS from elim step:

$$b_2 = b_2 - \frac{T_{21}}{T_{11}} b_1$$

- These two steps can be encoded generally as (j^{th} row)

1) $$T_{ij} = T_{ij} - \left(\frac{T_{i,i-1}}{T_{i-1,i-1}} \right) T_{i-1,j}$$

$$\left(i \in \{1, n\} \; ; \quad j \in \{i-1, i+1\} \right)$$

there are two of these to perform:

$$T_{i,i} = T_{i,i} - \left(\frac{T_{i,i-1}}{T_{i-1,i-1}} \right) T_{i-1,i}$$

$$T_{i,i+1} = T_{i,i+1} - \left(\frac{T_{i,i-1}}{T_{i-1,i-1}} \right) T_{i-1,i+1}$$

2) Following by RHS:

$$b_i = b_i - \left(\frac{T_{i,i-1}}{T_{i-1,i-1}} \right) b_{i-1}$$

- Following these two steps $\underline{\underline{T}}$ has been triangularized

$\underline{\underline{T}} \rightarrow \underline{\underline{T}}'$; we can now perform backsubstitution...

$$x_n = b_n / T_{nn} \; ; \quad x_i = \frac{1}{T_{ii}} \left(b_i - T_{i,i+1} x_{i+1} \right)$$

Condition Number

- Elimination methods can be sensitive to round-off error which propagates thru subsequent steps...
- One way to measure this sensitivity is thru the matrix condition number which is computed using norms
- A norm is a measure of magnitude & satisfies the constraints:

a) $\|\underline{\underline{A}}\| > 0$

b) $\|\underline{\underline{A}}\| = 0$ iff $\underline{\underline{A}} = \underline{\underline{0}}$

c) $\|\alpha \underline{\underline{A}}\| = |\alpha| \, \|\underline{\underline{A}}\|$

d) $\|\underline{\underline{A}} + \underline{\underline{B}}\| = \|\underline{\underline{A}}\| + \|\underline{\underline{B}}\|$

e) $\|\underline{\underline{A}}\,\underline{\underline{B}}\| \leq \|\underline{\underline{A}}\| \, \|\underline{\underline{B}}\|$ (Schwarz inequality)

- Commonly used vector norms:

$$\|\underline{x}\|_1 = \sum_i |x_i| \qquad L^1\text{-norm}$$

$$\|\underline{x}\|_2 = \sqrt{\sum_i x_i^2} \qquad L^2\text{-norm} \quad \text{(Euclidean)}$$

$$\|\underline{x}\|_\infty = \max(|x_i|) \qquad L^\infty\text{-norm}$$

- Matrix norms can be similarly defined:

$$\|\underline{\underline{A}}\|_1 = \max_j \left\{ \sum_i |A_{ij}| \right\} \quad \text{(max col. sum)}$$

$$\|\underline{\underline{A}}\|_\infty = \max_i \left\{ \sum_j A_{ij} \right\} \quad \text{(max row sum)}$$

$$\|\underline{\underline{A}}\|_2 = \min(\lambda_i) \quad \text{(min eigenvalue, spectral norm)}$$

$$\|\underline{\underline{A}}\|_e = \sqrt{\sum_{i,j} A_{ij}^2} \quad \text{(Euclidean norm)}$$

- The condition number measures sensitivity to small changes (e.g. round-off) in system.
- Considering the system:

$$\underline{\underline{A}}\,\underline{x} = \underline{b}$$

we know $\|\underline{b}\| \le \|\underline{\underline{A}}\|\,\|\underline{x}\|$

- Let us examine system perturbed by: $\underline{\delta b}$

$$\underline{\underline{A}}(\underline{x} + \underline{\delta x}) = \underline{b} + \underline{\delta b} = \underline{\underline{A}}\,\underline{x} + \underline{\underline{A}}\,\underline{\delta x} = \underline{b} + \underline{\delta b}$$

- Subtract original equation:

$$\underline{\underline{A}}\,\underline{\delta x} = \underline{\delta b}$$

$$\Rightarrow \quad \underline{\delta x} = \underline{\underline{A}}^{-1}\,\underline{\delta b}$$

thus $\|\underline{\delta x}\| \le \|\underline{\underline{A}}^{-1}\|\,\|\underline{\delta b}\|$

- Multiplying the two Schwarz-inequalities:

$$\|\underline{b}\|\,\|\underline{\delta x}\| \le \|\underline{\underline{A}}\|\|\underline{x}\|\,\|\underline{\underline{A}}^{-1}\|\,\|\underline{\delta b}\|$$

$$\|\underline{b}\|\,\|\underline{\delta x}\| \le \|\underline{x}\|\,\|\underline{\delta b}\|\left(\|\underline{\underline{A}}\|\,\|\underline{\underline{A}}^{-1}\|\right)$$

$$\frac{\|\underline{\delta x}\|}{\|\underline{x}\|} \le \frac{\|\underline{\delta b}\|}{\|\underline{b}\|}\, C(\underline{\underline{A}})$$

$$C(\underline{\underline{A}}) \equiv \|\underline{\underline{A}}\|\,\|\underline{\underline{A}}^{-1}\| \quad \text{(condition number)}$$

- A large condition number $\Rightarrow$ modest $\|\underline{\delta b}\|/\|\underline{b}\|$ can lead to large $\|\underline{\delta x}\|/\|\underline{x}\|$ (BAD!!)
- $C(\underline{\underline{A}}) \sim 1$ is best.

Iterative Methods

- One drawback of elimination methods is the accumulation of error through successive steps.
- Iterative methods can address this issue, but require a matrix to be <u>diagonally dominant</u>, i.e.

$$\exists i \quad \text{s.t.} \quad |A_{ii}| \geq \sum_{j,\, j \neq i} |A_{ij}|$$

- Many systems will not meet this requirement...
- Iterative methods start with an initial guess and then refine that guess repeatedly until <u>convergence</u> is achieved.
- Consider: $\underline{\underline{A}}\,\underline{x} = \underline{b}$, for some guess $\underline{x}^{(0)}$

  the <u>residual</u> (error in a sense) is $\underline{b} - \underline{\underline{A}}\,\underline{x}^{(0)}$

$$R_i^{(k)} = b_i - \sum_j A_{ij} x_j^{(k)}$$

residual on i^{th} var @ iteration k

- When the residual is small the iterated solution is near the true solution.
- When the solution $\underline{x}^{(k)}$ does not change appreciably between iterations, it has <u>converged</u>.

- Jacobi iteration involves a correction to each variable based on residual:

$$x_i^{(k+1)} = x_i^{(k)} + \frac{R_i^{(k)}}{A_{ii}}$$

- The basis of this idea is that, given some guess for all vars. (sans i) we can directly compute the variable i:

$$A_{ii} x_i^{(k+1)} \approx b_i - \sum_{j \neq i} A_{ij} x_j^{(k)}$$

$$= A_{ii} x_i^{(k)} + b_i - \sum_j A_{ij} x_j^{(k)}$$

$$\therefore \quad x_i^{(k+1)} = x_i^{(k)} = \frac{1}{A_{ii}} \left(b_i - \sum_j A_{ij} x_j^{(k)} \right)$$

- Letting $\underline{\Delta x} = \underline{x}^{(k+1)} - \underline{x}^{(k)}$ we can define the convergence as:

$$|\underline{\Delta x}| \leq \varepsilon \quad \text{for some iteration step}$$

- Relative error can also be used for convergence but becomes problematic if a solution is near zero.

- Gauss-Seidel iteration uses the partially updated solution to determine $R_i^{(k)}$:

$$R_i^{(k)} = b_i - \sum_{j<i} A_{ij} x_j^{(k+1)} - \sum_{j \geq i} A_{ij} x_j^{(k)}$$

$$x_i^{(k+1)} = x_i^{(k)} + \frac{R_i^{(k)}}{A_{ii}} \quad \text{(same)}$$

- Convergence is almost always faster than Jacobi iteration & it's "simpler" to code.
- Successive over-relaxation (SOR) is a further refinement to speed convergence.

$$x_i^{(k+1)} = x_i^{(k)} + \omega \frac{R_i^{(k)}}{A_{ii}} \qquad (\omega < 2)$$

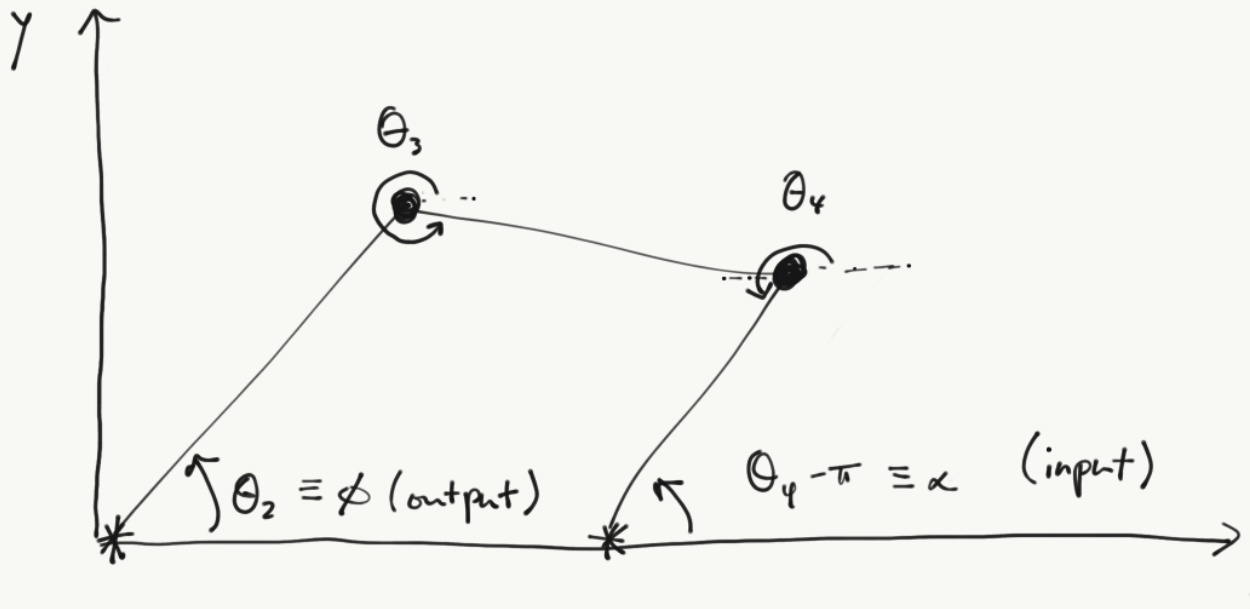
- The relaxation parameter (ω) must be chosen essentially by hand & some choices will make convergence slower...

Nonlinear Equations

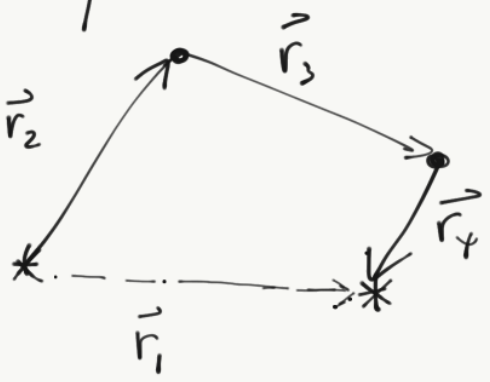
- Solutions of nonlinear equations are necessary in a wide range of applications.
- The canonical nonlinear problem is that of finding the roots of a function. Ie find

$$\{x_i\} \quad \text{s.t.} \quad f(x_i) = 0$$

- The book gives the example of a four bar linkage:



Vectorially we have:



$$\vec{r}_1 = \vec{r}_2 + \vec{r}_3 + \vec{r}_4$$

(x) $r_1 = r_2\cos\theta_2 + r_3\cos\theta_3 + r_4\cos\theta_4$

(y) $0 = r_2\sin\theta_2 + r_3\sin\theta_3 + r_4\sin\theta_4$

- Nonlinear equations can be extremely difficult to solve by hand, e.g.:

$$x - e^{-x} = 0 \qquad \text{(transcendental)}$$

- Often you just have to plot things or start guessing...

Root finding

- Root finding procedures (finding zeros of $f(x)$) generally share several basic steps:

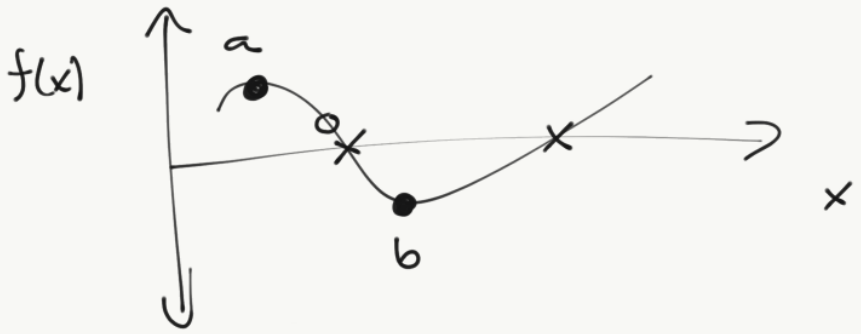
  1) bounding (restriction of solution to some interval, $a \leq x_i \leq b$)
  2) refinement (update solution to reduce error)

- Bounding
  a) graphing or search
  b) prior knowledge
  c) solution of approximate model
  d) prior solution in sequence of iterations.

- Refinement
  a) guessing
  b) "closed" domain methods
  c) open domain "

- See figure 3.6 & 3.2.4
- Root finding algorithm features
  a) max # of iterations
  b) error checking on whether $f'(x) \to 0$
  c) convergence test (x_i not changing or $f(x_i)$ small)
  d) checking that the root actually works.

- Interval halving is a straightforward way to implement a search method for root-finding
- If the value of the function a two different points changes sign, & that function is continuous than the root lies between those points:



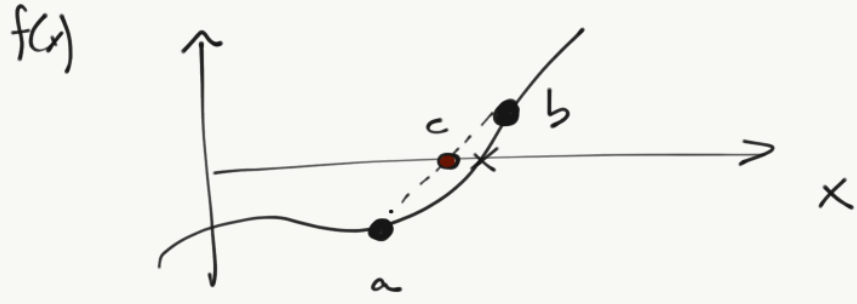
- Just guessing one could estimate that the root is halfway in between

$$x_i \sim \frac{a+b}{2}$$

- We can then choose the half interval (a,c) or $(c,b]$ that contains the root (fn. changes sign) & iterate until $f(c) = 0$ to some desired tolerance.

- False position is a refinement of interval halving.

- Rather than taking c @ the midpoint of (a,b), we approximate a line connecting $f(a)$ & $f(b)$ & find where that line crosses zero to estimate c.



(value of fn. at left ↑ "boundary")

- The line is defined by $y = m(x-c) + y_a$

$$m \approx \frac{f(b)-f(a)}{b-a} \; ; \quad y(x) = \frac{f(b)-f(a)}{b-a}(x-a) - f(a)$$

$$c: \qquad 0 = \frac{f(b)-f(a)}{b-a}(c-a) + f(a)$$

↑

(position @ which $y(x) = 0$) ∴

$$c = a - f(a) \Big/ \frac{(b-a)}{f(b)-f(a)}$$

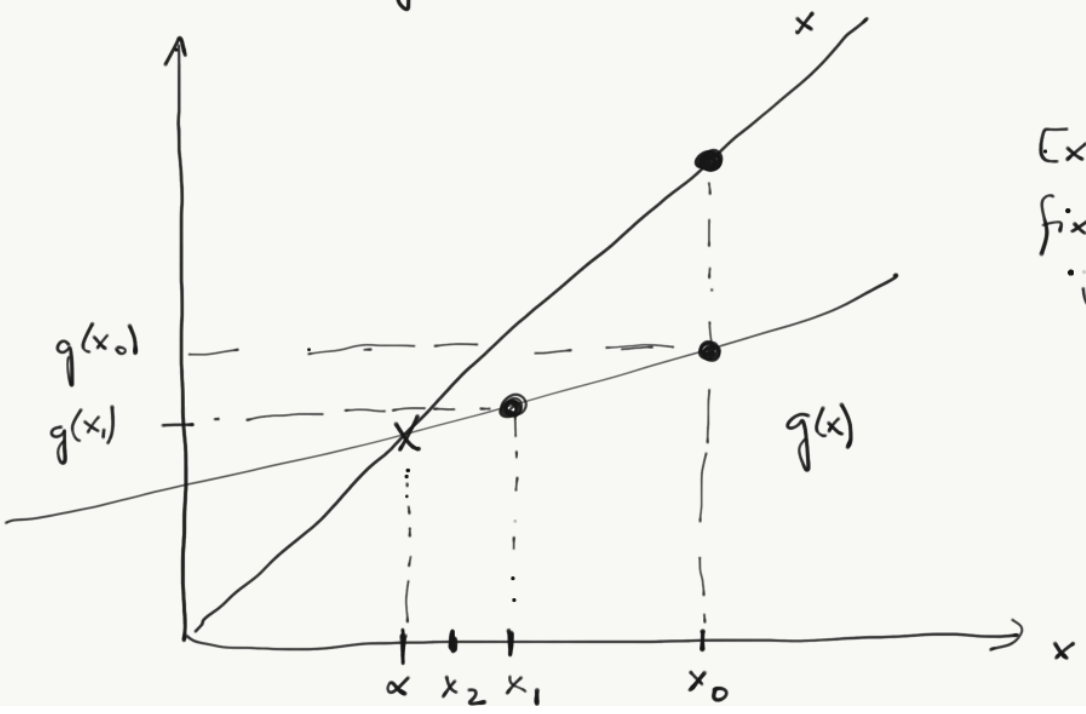
- Now as before we choose a new interval containing the root & iterate to convergence.

Fixed-point iteration

- Open domain methods do not require a root to be "bracketed"
  a) fixed-point iteration
  b) Newton's method
  c) secant method
- Fixed point iteration:
  1) recast $f(x) = 0$ into $x = g(x)$
  2) guess x_0 (initial root estimate)
  3) iterate to convergence:

$$x_{i+1} = g(x_i)$$

- This is similar in concept to Jacobi iterations for linear equations - guess solution & use it to solve remaining vars.



Ex. of fixed-point iteration

- Iterations (of any method) continue until some convergence is achieved.

$$|x_{i+1} - x_i| \le \varepsilon \quad \underline{\text{or}} \quad |f(x_{i+1})| \le \varepsilon'$$

- Suppose the true root (unknown) is denoted by α; the error is then $x - \alpha$.

$$x_{i+1} = g(x_i) \quad \therefore \quad x_{i+1} - \alpha = g(x_i) - \alpha$$

- At the solution $\alpha = g(\alpha)$:

$$x_{i+1} - \alpha = g(x_i) - g(\alpha)$$

thus: $$e_{i+1} = g(x_i) - g(\alpha)$$

- let us expand $g(\alpha)$ in a Taylor Series: (about x_i)

$$g(\alpha) = g(x_i) + g'(x_i)(\alpha - x_i) + \dots$$

then: $$e_{i+1} \approx g(x_i) - \{ g(x_i) + g'(x_i)(\alpha - x_i) \}$$

$$\approx -g'(x_i) \underbrace{(\alpha - x_i)}_{e_i}$$

(linear convergence: e_{i+1} is linear fn. of e_i)

$$\Rightarrow \boxed{e_{i+1} = g'(x_i)\, e_i} \Rightarrow |e_{i+1}| = |g'(x_i)\, e_i|$$

$$\boxed{|e_{i+1}| / |e_i| = |g'(x_i)| \quad (\text{must be} < 1)}$$

Newton's Method

- Converges quadratically, but requires derivative eval.

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$ (truncated Taylor series)

- Solving for x_{i+1}, we find:

$$f'(x_i)(x_{i+1} - x_i) = f_{i+1} - f_i$$

$$f'_i x_{i+1} = f_{i+1} - f_i + f'_i x_i$$

$$x_{i+1} = \frac{f_{i+1} - f_i}{f_i'} + x_i$$

- The purpose of an iteration like this is to drive the root to zero, viz. $f_{i+1} = 0$

$\therefore$

$$\boxed{x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}}$$

Newton method!

- Convergence can be quantified by noting similarity to fixed-point iteration:

$$x_{i+1} = g(x_i) \quad \text{w/} \quad g(x_i) = x_i - \frac{f(x_i)}{f'(x_i)}$$

- For fixed point $|g'(x_i)| < 1$ converges.

$$\frac{d}{dx}\left(x - \frac{f(x)}{f'(x)} \right) = 1 - \frac{f'f' - f''f}{(f')^2}$$

$$g'(x) = \frac{\cancel{(f')^2} - \cancel{(f')^2} + f''f}{(f')^2} = \frac{f''f}{(f')^2}$$

- Evaluating @ root $x = \alpha$ → (goes to zero @ $x=\alpha$)

$$g'(\alpha) = \frac{f''(\alpha) f(\alpha)}{(f')^2} = 0 \quad \text{(convergent!)}$$

- Similar to before $e = x - \alpha$

$$e_{i+1} = x_{i+1} - \alpha = x_i - \frac{f_i}{f_i'} - \alpha$$

$$= x_i - \alpha - f_i / f_i' = e_i - f_i / f_i'$$

- Expanding about x_i:

$$f(\alpha) = f(x_i) + f'(x_i)(\alpha - x_i) + \frac{1}{2} f''(x_i)(\alpha - x_i)^2 + \ldots$$

- At root $f(\alpha) = 0 \Rightarrow$

$$f(x_i) = f'(x_i)(x_i - \alpha) - \frac{1}{2} f''(x_i)(x_i - \alpha)^2$$

$$= f'(x_i) e_i - \frac{1}{2} f''(x_i) e_i^2$$

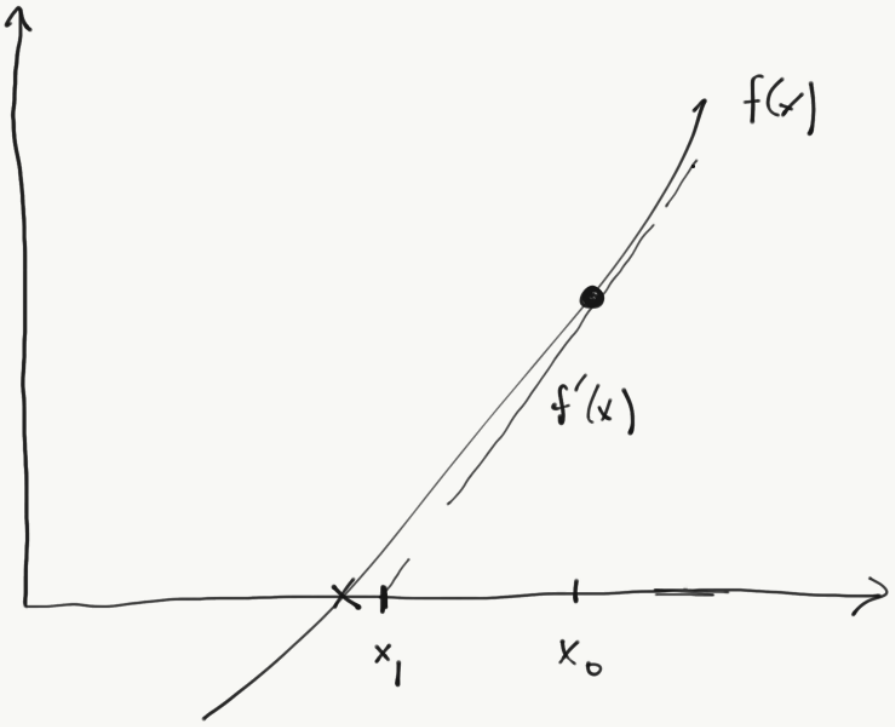
$$\therefore \quad e_{i+1} = e_i - \frac{f'(x_i) e_i - \frac{1}{2} f''(x_i) e_i^2}{f'(x_i)} = \frac{1}{2} \frac{f''(x_i)}{f'(x_i)} e_i^2$$

- In the case that derivatives are too expensive to evaluate they can be approximated:

$$\tilde{f}'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{(x_{i+1} - x_i)}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{\tilde{f}'(x_i)}$$ (approx. Newton's method)

Ex. of Newton's method



Secant Method

- Bears a lot of similarity to approximate Newton's method
- Given two initial approximations of the root (x_0, x_1), we take:

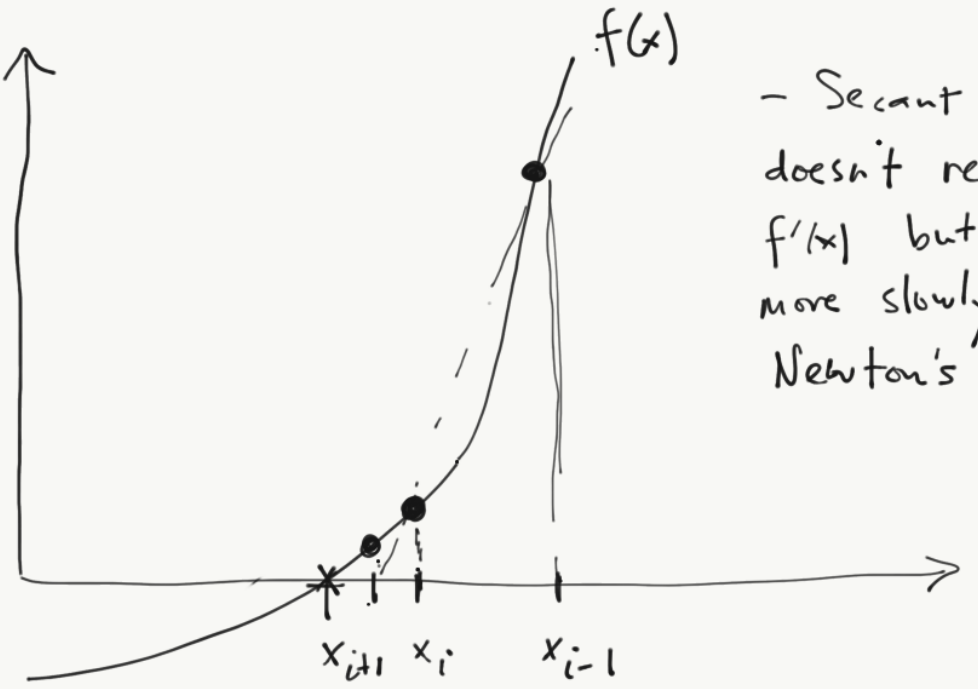
$$g'(x_i) \approx f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

- This derivative is used w/ Newton-style iteration:

$$x_{i+1} = x_i - \frac{f(x_i)}{g'(x_i)}$$

- This is equivalent to finding root of secant line as next approx. to root of $f(x)$.

(Ex. of secant method)



- Secant method doesn't require $f'(x)$ but converges more slowly than Newton's

Muller's Method

- Muller's method takes 3 initial points & approximates next root iteration from a quadratic form fitting these.
- Given x_i, x_{i-1}, x_{i-2} we find x_{i+1} from:

$$g(x) = a(x - x_i)^2 + b(x - x_i) + c$$

- With a, b, c determined from enforcing $g(x)$ passes thru f @ x_i, x_{i-1}, x_{i-2}

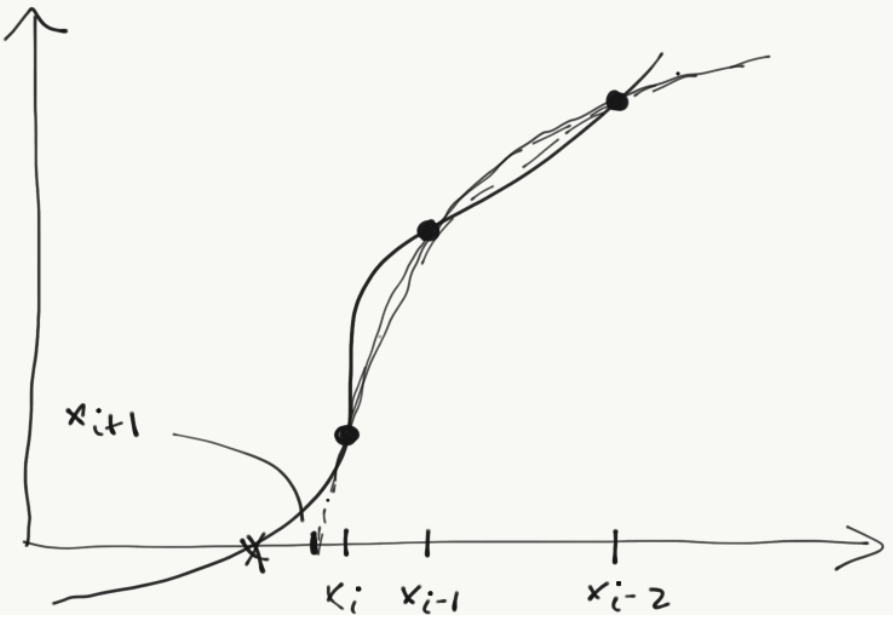
$$f_i = c$$

$$f_{i-1} = a(x_{i-1} - x_i)^2 + b(x_{i-1} - x_i) + c$$

$$f_{i-2} = a(x_{i-2} - x_i)^2 + b(x_{i-2} - x_i) + c$$

- a, b solved using latter two eqns. (see book)

Ex: Muller's method.



- Once a, b, c known, we find root of quadratic to estimate next value of root of $f(x)$:

$$x_{i+1} = x_i - \frac{2c}{b \pm \sqrt{b^2 - 4ac}}$$

- We choose root having $(\pm) \rightarrow \text{sign}(b)$ to prevent overly large changes $\Delta x = x_{i+1} - x_i$

Polynomial roots

(contains material from 4.2)

- Polynomials are of the form:

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n$$

- Fundamental thm of algebra: n^{th} degree polynomial has n roots.
- For linear & quadratics eqns can solve roots directly:

$$P_1(x) = ax + b \Rightarrow x = -b/a$$

$$P_2(x) = ax^2 + bx + c \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

if roots have Im part they occur in complex conjugate pairs.

- Descartes' rule of signs: number of positive roots of $P_n(x)$ is number of sign changes in nonzero coeffs or smaller by an even integer. Number of negative roots found by considering $P_n(-x)$ in the same manner.
- Root-finding tends to be ill-conditioned in that small changes to P_n coeffs can greatly change roots.
- Polynomial evaluations require lots of multiplications due to x^2, x^3, x^4 and so on.

- Nested multiplication can reduce number of operations needed to evaluate polynomial e.g.

$$P_4(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$$

$$= a_0 + x(a_1 + x(a_2 + x(a_3 + x a_4)))$$

- More generally, nested multiplication goes like:

$$P_n(x) = a_0 + x(a_1 + x(a_2 + \dots x(a_{n-1} + x a_n)\dots))$$

which can be evaluated thru the sequence:

$$b_n = a_n$$
$$b_i = a_i + x b_{i+1} \quad (i = n-1, 0)$$

$$\Rightarrow \underline{b_0 = P_n(x)}$$

- Polynomial division is defined by:

$$P_n(x) = (x-N)\, Q_{n-1}(x) + R$$

(N: divisor; $Q_{n-1}(x)$: quotient (degree $n-1$); R: remainder)

also note that, generally

$$P_n(N) = R$$

note $P_n(N) = 0 \Rightarrow (x-N)$ is a factor of $P_n(x)$
$\Rightarrow N$ is a root

- From poly-divison we have the following derivative constraint:

$$P_n'(x) = Q_{n-1}(x) + (x-N)\,Q_{n-1}'(x)$$

- Therefore, at the root N:

$$P_n'(N) = Q_{n-1}(N)$$

$$Q_{n-1} = \sum_{i=1}^{n-1} b_i x^{i-1}$$

$$P_n(x) = \sum_{i=0}^{n} a_i x^i$$

- Given a number, N, we can perform divison as follows:

$$P_n(x) = (x-N)\,Q_{n-1}(x) + R$$

$$\sum_{i=0}^{n} a_i x^i = (x-N) \sum_{i=1}^{n-1} b_i x^{i-1} + R$$

$$\sum_{i=0}^{n} a_i x^i = \sum_{i=1}^{n-1} b_i x^i - \left(\sum_{i=1}^{n-1} b_i x^{i-1} \right) N + R$$

- Polynomial representations are unique so we can equate coefficients of the powers:

$$i' = i-1$$
$$i = i'+1$$
$$i = 1 \Rightarrow i' = 0$$
$$i = n-1 \Rightarrow i' = n-2$$

$$\sum_{i=0}^{n} a_i x^i = \sum_{i=1}^{n-1} b_i x^i - N \left(\sum_{i'=0}^{n-2} b_{i'+1} x^{i'} \right) + R$$

$$a_i = b_i - N b_{i+1} \Rightarrow b_i = a_i + N b_{i+1}$$

- First & last terms are special cases:

$$b_n = a_n$$

$$b_0 \equiv R = a_0 + N b_1$$

- So synthetic division can be summarized as:

$$b_n = a$$

$$b_i = a_i + N b_{i+1} \quad (i = n-1, 1)$$

$$R = a_0 + N b_1$$

Newton's method for polynomials

- Newton's method works normally for simple (i.e non repeated) roots.
- For polynomials it can be desirable to <u>deflate</u> the polynomial once a root is found

$$P_n(x) = (x - \alpha_1) Q_{n-1}(x)$$

- The remaining roots are then those of $Q_{n-1}(x)$
- The roots off $Q_2(x)$ can be found via quadratic formula. This offers a straight forward way to converge on all roots...
- Multiple roots are handled fine via Newton's method & there a some algorithms to speed convergence (see 3.5.2.3.
  a) multiplicity factor
  b) modified function in Newton's method
- For multiple roots need to insure $f'(x) \neq 0 \Rightarrow$ use a relative convergence criteria based on $\Delta x = x_{i+1} - x_i$

- For complex roots Newton's method can converge if an initial complex guess is chosen. (Bracketing methods do not work)

- Often deflated root solutions are refined through an additional iteration step.

Systems of nonlinear equations

- Newton's method can readily be adapted to solve multi-variate problems, e.g.:

$$f(x,y) = 0 \quad ; \quad g(x,y) = 0$$

- Expanding in a two-variable Taylor series:

$$f(x,y) = f(x_i,y_i) + \frac{\partial f}{\partial x}(x_i,y_i)(x-x_i) + \frac{\partial f}{\partial y}(x_i,y_i)(y-y_i) + \dots$$

$$g(x,y) = g_i + \left[\frac{\partial g}{\partial x}\right]_i (x-x_i) + \left[\frac{\partial g}{\partial y}\right]_i (y-y_i) + \dots$$

- To solve this system, we wish to drive f, g to zero $\Rightarrow f(x,y), g(x,y) \to 0$.

$$\begin{bmatrix} (\partial f/\partial x)_i & (\partial f/\partial y)_i \\ (\partial g/\partial x)_i & (\partial g/\partial y)_i \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} -f_i \\ -g_i \end{bmatrix}$$

followed by the simple update $\quad x_{i+1} = x_i + \Delta x$

$$y_{i+1} = y_i + \Delta y$$

- More generally, for n equations, n unknowns:

$$f_j(x_k) = 0$$

$\{x_k\}$: unknowns

$\{f_j\}$: nonlinear equations.

- Define a matrix:

$$D_{jk} = \frac{\partial f_j}{\partial x_k}$$

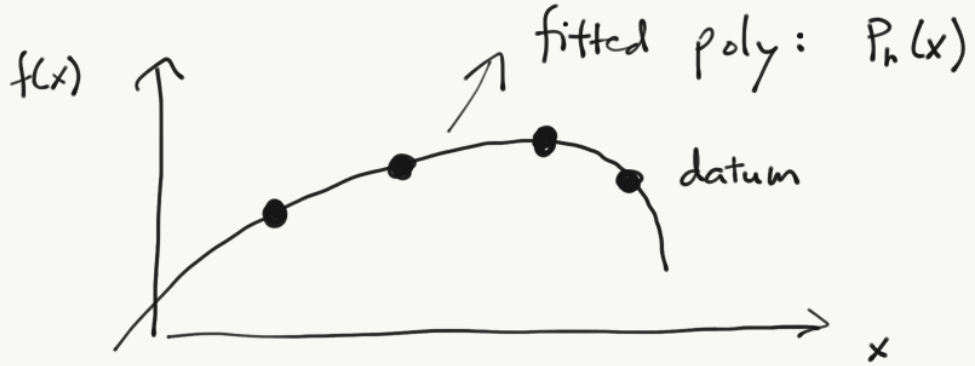
$$\underline{\underline{D}}\,\underline{\Delta x} = -\underline{f}$$

$$\underline{x}_{it1} = \underline{x}_i + \underline{\Delta x}$$

} So @ each iteration a linear system must be solved!

Polynomials & fitting

- The most common applications polynomials is for fitting data or performing interpolation.
- They are easy to differentiate & integrate & inexpensive to compute
- Fitting can be either <u>exact</u> (passing thru all available data) or approximate (minimizing some error or residual metric)
- Most straightforward use of polynomials is <u>direct fitting</u>: viz. find the n^{th} degree polynomial exactly passing through n+1 points.



$$\underbrace{P_n(x)}_{f(x)} = a_0 + a_1 x + \dots a_n x^n = \sum_{i=1}^{n} a_i x^i$$

$$f_j = \sum_{i=1}^{n} a_i x^i \rightsquigarrow (\underline{x}^n)^T \underline{a} \quad \text{(matrix mult.)}$$

$\underline{x}^n$: powers of x; $\underline{a}$: coeff vector

$$\underline{f} = \underline{\underline{X}}\,\underline{a} \longrightarrow \text{solve for } \underline{a} \qquad \underline{\underline{X}} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots \\ 1 & x_2 & x_2^2 & \\ 1 & x_3 & x_3^2 & \\ \vdots & & & \end{bmatrix}$$

Approximate fits

- In some situations we have effectively more data than unknowns such that the system is <u>over-determined</u>.

  under-determined: fewer eqns. than unknowns (need prior info to solve)
  even-determined: same # of eqns. & unknowns
  over-determined: more eqns. than unknowns.

- For an over determined problem we seek to fit the data in the <u>best possible way</u>. Letting Y_i be the data & $f(x)$ over objective fn.

$$e_i = Y_i - f(x_i) \qquad \text{(error / deviation)}$$

- least squares minimizes squared error: e_i^2
- linear objective fns. often used:

$$f(x) = a + bx \;;\; \text{need to } \underline{\text{minimize}} \quad S = \sum_i (Y_i - f(x_i))^2$$

where S is understood to be a fn. of a, b.

1) $\partial S/\partial a = \sum_i 2(Y_i - f(x_i))(-1)$
2) $\partial S/\partial b = \sum_i 2(Y_i - f(x_i))(-x_i)$

} <u>set to zero for extrema</u>

1) $\sum_i Y_i = \sum_i a + bx_i$ 2) $\sum_i x_i Y_i = \sum_i (ax_i + bx_i^2)$

- Note that there are just two unknowns to solve for:

1) $a \sum_i 1 + b \sum_i x_i = \sum_i y_i$

2) $a \sum_i x_i + b \sum_i x_i^2 = \sum_i x_i y_i$

$$\underline{\underline{M}}\,\underline{a} = \underline{y}'$$

$$\underline{\underline{M}} = \begin{bmatrix} \sum_i 1 & \sum_i x_i \\ \sum_i x_i & \sum_i x_i^2 \end{bmatrix} \qquad \underline{a} = \begin{bmatrix} a \\ b \end{bmatrix}$$

$$\underline{y}' = \begin{bmatrix} \sum_i y_i \\ \sum_i x_i y_i \end{bmatrix}$$

(matrix system to be solved) ...

- Note that the original system of equations can be viewed itself in the context of matrix multiplications:

$$f(x_i) = a + b x_i \rightsquigarrow f_i = \underline{x}_i^T \underline{a}$$

$$\underline{x}_i^T = \begin{bmatrix} 1 & x_i \end{bmatrix} \qquad \underline{a}^T = \begin{bmatrix} a & b \end{bmatrix}$$

For the entire set of points: $\underline{f} = \underline{\underline{X}}\,\underline{a}$

$$\underline{\underline{X}} = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} ; \quad \underline{f} = [f_i] ; \quad \underline{a}^T = \begin{bmatrix} a & b \end{bmatrix}$$

- Note that, due to linearity of f we may write:

$$f_i = \begin{bmatrix} \partial f/\partial a & \partial f/\partial b \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

(Jacobian matrix)

$$\underline{f} = \begin{bmatrix} \left.\frac{\partial f}{\partial a}\right|_1 & \left.\frac{\partial f}{\partial b}\right|_1 \\ \vdots & \vdots \\ \left.\partial f/\partial a\right|_n & \left.\partial f/\partial b\right|_n \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \underline{\underline{J}}\,\underline{q}$$

- Because this is not a square system there is no direct solution for $\underline{a}$
- The error is:

$$\underline{e} = \{e_i\} = \underline{Y} - \underline{f} \quad \text{(vector of errors)}$$

- The total squared error $\sum_i e_i^2$ is then:

$$\underline{e}^T\underline{e} = (\underline{Y} - \underline{\underline{J}}\,\underline{q})^T(\underline{Y} - \underline{\underline{J}}\,\underline{a}) = \sum_i e_i^2$$

- And the system of eqns. to be solved is:

$$\frac{\partial}{\partial a_j}(\underline{e}^T\underline{e}) = 0 \qquad (j \text{ runs over unknowns } (1,2))$$

i: # of data points j: # of unknowns.

$$\frac{\partial}{\partial a_j}(\underline{e}^T\underline{e}) = \frac{\partial}{\partial a_j}(\underline{e}^T)\,e + \underline{e}^T\frac{\partial \underline{e}}{\partial a_j}$$

note: $\dfrac{\partial \underline{e}}{\partial a_j} = -\underline{J}_j$ (jth column of $\underline{\underline{J}}$ times a_j)

- Similarly: $\frac{\partial \underline{e}^T}{\partial a_j} = \underline{J}_{:j}^T$

$\underline{a} \sim m \times 1 \quad \underline{Y} \sim n \times 1$

$\underline{e} \sim n \times 1 \quad \underline{\underline{J}} \sim n \times m$

$$\frac{\partial}{\partial a_j}(\underline{e}^T \underline{e}) = \underline{J}_{:j}^T \underline{e} + \underline{e}^T (\underline{J}_{:j})$$

conform to add!!!

- The full system to be solved is given by:

$m \times 1$

$$\frac{\partial}{\partial \underline{a}}(\underline{e}^T \underline{e}) = (\underline{\underline{J}}^T)\underline{e} + (\underline{e}^T(\underline{\underline{J}}))^T = 0$$

↳ vector derivative, viz. gradient

$$\underline{\underline{J}}^T(\underline{Y} - \underline{\underline{J}}\underline{a}) + ((\underline{Y} - \underline{\underline{J}}\underline{a})^T \underline{\underline{J}})^T = 0$$

$$\underline{\underline{J}}^T \underline{Y} - \underline{\underline{J}}^T \underline{\underline{J}}\underline{a} + \underline{\underline{J}}^T(\underline{Y} - \underline{\underline{J}}\underline{a}) = 0$$

$$\underline{\underline{J}}^T \underline{Y} - \underline{\underline{J}}^T \underline{\underline{J}}\underline{a} + \underline{\underline{J}}^T \underline{Y} - \underline{\underline{J}}^T \underline{\underline{J}}\underline{a} = 0$$

$$2(\underline{\underline{J}}^T \underline{\underline{J}})\underline{a} = 2\underline{\underline{J}}^T \underline{Y}$$

$$(\underline{\underline{J}}^T \underline{\underline{J}})\underline{a} = \underline{\underline{J}}^T \underline{Y}$$

$$\therefore \quad \underline{a} = \underbrace{(\underline{\underline{J}}^T \underline{\underline{J}})^{-1} \underline{\underline{J}}^T}\ \underline{Y}$$

Moore-Penrose inverse

(solves over determined systems)

Lagrange Polynomials

- Given two points $[a, f(a)]$ $[b, f(b)]$ the line passing thru these is given by:

$$P_1(x) = \underbrace{\frac{(x-b)}{(a-b)} f(a)}_{0 \text{ for } x=b} + \underbrace{\frac{(x-a)}{(b-a)} f(b)}_{0 \text{ for } x=a}$$

- Note that:

$$P_1(a) = \frac{(a-b)}{(a-b)} f(a) = f(a) \quad \checkmark$$

$$P_1(b) = f(b) \quad \checkmark$$

- Consider three points & the quadratic form:

$$a, b, c \rightarrow f(a), f(b), f(c)$$

$$P_2(x) = \frac{(x-b)(x-c)}{(a-b)(a-c)} f(a) + \frac{(x-a)(x-c)}{(b-a)(b-c)} f(b) + \frac{(x-a)(x-b)}{(c-a)(c-b)} f(c)$$

- This polynomial has similar properties: (1) coeff of $f(a)$ is zero for $x = b$ or c, (2) coeff. of $f(a) = 1$ for $x = a$.

- Extension to higher order polynomial:

$$P_n(x) = \sum_{j=1}^{n+1} \frac{\prod_{i \neq j} (x - x_i)}{\prod_{i \neq j} (x_j - x_i)} f(x_j)$$

NOTE: poly representations are unique so that this gives same result as direct fit!

- lagrange polynomials can be constructed sequentially thru use of differences:

$$f(x) = \frac{(x-b)}{(a-b)} f(a) + \frac{(x-a)}{(b-a)} f(b)$$

$$= \frac{f(b)(x-a) - f(a)(x-b)}{(b-a)}$$

- For two points $[a,b] \rightarrow [x_i, x_{i+1}]$

$$f_i^{(1)} = \frac{f(x_{i+1})(x-x_i) - f(x_i)(x-x_{i+1})}{(x_{i+1} - x_i)}$$

- For three points $[x_i, x_{i+1}, x_{i+2}]$:

$$f_i^{(2)} = \frac{(x-x_i) f_{i+1}^{(1)} - (x-x_{i+2}) f_i^{(1)}}{x_{i+2} - x_i}$$

$$f_i^{(n)} = \frac{(x-x_i) f_{i+1}^{(n-1)} - (x-x_{i+n}) f_i^{(n-1)}}{x_{i+n} - x_i}$$

generally, this can be most easily seen by direct substitution

- Computing coefficients this way is known as Neville's Algorithm

- Divided differences & associated polynomials are another way of conceptualizing Lagrange polynomials:

$$f_i^{(1)} = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} \quad \text{(derivative approximation)}$$

$$f_i^{(2)} = \frac{f_{i+1}^{(1)} - f_i^{(1)}}{x_{i+2} - x_i} \quad (2^{\text{nd}} \text{ derivative})$$

$$= \frac{\dfrac{f_{i+2} - f_{i+1}}{x_{i+2} - x_{i+1}} - \dfrac{f_{i+1} - f_i}{x_{i+1} - x_i}}{x_{i+2} - x_i}$$

$$= f_{i+2}\left(\frac{1}{(x_{i+2} - x_{i+1})(x_{i+2} - x_i)}\right)$$

$$+ f_{i+1}\left(\frac{-1}{(x_{i+2} - x_{i+1})(x_{i+2} - x_i)} - \frac{1}{(x_{i+1} - x_i)(x_{i+2} - x_i)}\right)$$

$$+ f_i\left(\frac{1}{(x_{i+1} - x_i)(x_{i+2} - x_i)}\right)$$

- These can also be used to construct polynomials that pass thru $n+1$ pts. (→ of n^{th} degree)

$$f(x) = f_i^{(0)} + f_i^{(1)}(x - x_i) + f_i^{(2)}(x - x_{i+1})(x - x_i) + \ldots$$
$$f_i^{(n)}(x - x_{i+n-1})(x - x_{i+n-2}) \cdots (x - x_i)$$

(Finite) Differences

- Forward difference : $\Delta f_i \equiv (f_{i+1} - f_i)$
- Backward diff : $\nabla f_i = (f_i - f_{i-1})$
- Centered diff : $\delta f_i = (f_{i+1/2} - f_{i-1/2})$

OR $\delta f_{i+1/2} = (f_{i+1} - f_i)$

- The book makes frequent use of difference tables:

x	$f(x)$	Δf	$\Delta^2 f$	$\Delta^3 f$
x_0	f_0	Δf_0		
x_1	f_1		$\Delta^2 f_0$	
		Δf_1		$\Delta^3 f_0$
x_2	f_2		$\Delta^2 f_1$	
		Δf_2		
x_3	f_3			

- Higher order differences are defined by:

$$\Delta^n f_i = \Delta^{(n-1)} f_{i+1} - \Delta^{(n-1)} f_i$$

So that $\Delta^2 f_i = \Delta f_{i+1} - \Delta f_i$

$$= f_{i+2} - f_{i+1} - (f_{i+1} - f_i)$$

$$= f_{i+2} - 2f_{i+1} + f_i$$

and so on ...

- Note that errors get compounded as one goes across the table an computes higher order differences.
- The Newton forward difference polynomial is defined by:

$$P_n(x) = f_0 + s\,\Delta f_0 + \frac{s(s-1)}{2!}\Delta^2 f_0 + \frac{s(s-1)(s-2)}{3!}\Delta^3 f_0 + \dots + \frac{s(s-1)(s-2)\dots(s-(n-1))}{n!}\Delta^n f_0$$

where $s = \frac{x - x_0}{\Delta x} = \frac{x - x_0}{h} \Rightarrow x = x_0 + sh$

$(s \in \mathbb{Z}^+)$ (set of positive integers)

- Coefficients of this polynomial are binomial coefficients:

$$\frac{s(s-1)(s-2)\dots(s-(n-1))}{n!} = \frac{s!}{n!\,(s-n)(s-n-1)\dots(1)}$$

$$= \boxed{\frac{s!}{n!(s-n)!} \equiv \binom{s}{n}} \quad \text{"s choose n"}$$

- Other forms of difference polynomials also exist (see book).

Multivariate Approximation

- We are often faced with fitting or approximation of functions of more than one variable.

e.g. $z = f(x,y)$

usually a multivariate polynomial is a good choice:

$$f(x,y) = a + bx + cy + dxy + ex^2 + fy^2$$
$$+ gx^2y + hy^2x + \ldots$$

- A direct fit approach can work for lower-degree functions $f(x,y)$: (bilinear interpolant)

$$f(x,y) \approx a + bx + cy + dxy$$

- For a given set of four data points we can produce a solution for $\{a, b, c, d\}$

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 \\ 1 & x_2 & y_2 & x_2y_2 \\ 1 & x_3 & y_3 & x_3y_3 \\ 1 & x_4 & y_4 & x_4y_4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

which can be solved thru standard linear algebraic approaches.

- Solution for higher-order coefficients will be ill-conditioned.
- Successive univariate approximation is often a better approach:

$$f_{j*}(x) = f(x, y_j) \approx a_j + b_j x + c_j x^2 + d_j x^3 + \dots$$

denote: $f_{ij} \Rightarrow f(x_i, y_j)$ (one of these for each y_j point)

$$\begin{bmatrix} f_{1j} \\ f_{2j} \\ f_{3j} \\ \vdots \\ f_{nj} \end{bmatrix} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & & x_2^{n-1} \\ 1 & x_3 & & x_3^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} a_i \\ b_i \\ c_i \\ \vdots \end{bmatrix}$$

which can be solved for the coefficients a_j, b_j etc

- Once we have interpolating fns. for $f(x, y_j)$ we can fit these to polynomials in the y variable:

$$f(x_i, y) = a_i' + b_i' y + c_i' y^2 + \dots$$

↳ evaluated using $f_{j*}(x)$, then set up a system to solve:

approximating fn. is given from $\underline{a}'$

$$\underline{f}_* = \underline{\underline{Y}}\, \underline{a}' \rightarrow \boxed{\underline{a}' = \underline{\underline{Y}}^{-1} \cdot \underline{f}_x}$$

(probably use Gauss elim)

Splines

- Splines are effectively piecewise polynomials fitted to a subset of available data points.
- Cubics are the most commonly used as their slope & curvature can be forced to be continuous @ each data point.

$$f_i(x) = a_i + b_i x + c_i x^2 + d_i x^3 \qquad (i = 1, n+1)$$

($n+1$: total # of data pts.)

- Cubic Splines satisfy the constraints:

0) $f_i(x)$ must pass through the data point @ $x = x_i$: $n-1$ constraints. (interior)

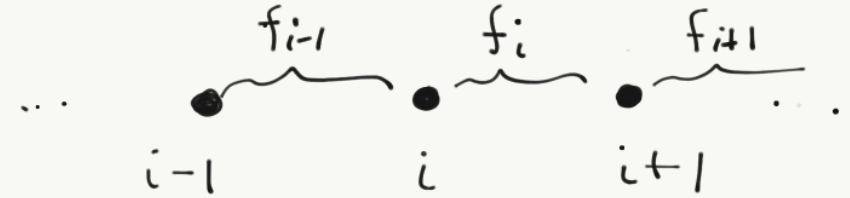
1) $f_i(x)$ & $f_{i+1}(x)$ have the same value @ $x = x_i$: $(n-1)$ constraints.

2) 1st derivatives are equal on each side of a given pt. (interior) : $f_i'(x) = f_{i+1}'(x)$ @ x_{i+1}

$(n-1)$ equations

3) 2nd derivatives of adjacent splines are equal @ interior grid points : $f_i''(x) = f_{i+1}''(x)$ @ x_{i+1}

$(n-1)$ equations



$n-1$ interior intervals

(STENCIL)

- There remains four additional constraints to specify a unique solution.

4) f_1 must pass thru the data @ x_1 & f_n must pass thru the data @ x_{n+1} : 2 equations

5) curvature is specified @ the first & last grid points : 2 equations

Numerical Differentiation

- One approach to differentiate a function is to perform differentiation on an approximation (like a polynomial):

$$\frac{df}{dx} = \frac{d}{dx}\left(P_n(x)\right)$$

- Integration can similarly be performed in this manner

$$\int f(x)\,dx = \int P_n(x)\,dx$$

- For differentiation it is often most enlightening to use a Taylor Series approach:

$$f(x) = \sum_n f^{(n)}(x_0)\frac{(x-x_0)^n}{n!}$$

$$= f(x_0) + f'(x_0)(x-x_0) + \frac{f''(x_0)}{2!}(x-x_0)^2 + \dots$$

- Letting $x = x_0 + \Delta x$

(a) $$f(x+\Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{f''(x_0)}{2!}\Delta x^2 + \frac{f'''}{3!}\Delta x^3$$

- Similarly expanding for $x - \Delta x$:

(b) $$f(x-\Delta x) = f(x_0) - f'(x_0)\Delta x + \frac{f''}{2!}\Delta x^2 - \frac{f'''}{3!}\Delta x^3 + \dots$$

- These may be combined (a - b):

$$f(x+\Delta x) - f(x-\Delta x) = 2f'(x_0)\Delta x + \frac{2}{3!}f'''(x_0)\Delta x^3$$

$$f'(x_0) = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} + \frac{1}{6} f'''(x_0)\,\Delta x^2 \qquad (2)$$

$$= \quad '' \quad + \underbrace{\mathcal{O}(\Delta x^2)}_{\text{error term varying like } \Delta x^2}$$

- These two eqns. (a, b) can be added to yield an approx. 2nd derivative:

$$f(x_0+\Delta x) + f(x_0-\Delta x) = 2f(x_0) + f''(x_0)\,\Delta x^2 + \frac{1}{4!} f''''(x_-)\,\Delta x^4 + \dots$$

$$\therefore \quad f''(x_0) \approx \frac{f(x_0+\Delta x) - 2f(x_0) + f(x-\Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x^2)$$

- Often use indices to represent locations on a grid

$$x_0 \to i \;;\; x_0+\Delta x \to i+1 \;;\; x_0-\Delta x \to i-1$$

$$\left[\frac{df}{dx}\right]_i = \frac{f_{i+1} - f_{i-1}}{2\Delta x}$$

$$\left[\frac{d^2f}{dx^2}\right]_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2}$$

for n^{th} order derivative generally need to solve n equations from Taylor Series

→ (stencil of FD approx.)

- Less accurate approximations are possible:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''}{2!}(x - x_0)^2 + \dots$$

$$f(x + \Delta x) = f(x_0) + f'(x_0)\Delta x + \frac{f''}{2!}\Delta x^2 + \dots$$

$$f'(x_0) = \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x} + \frac{f''}{2!}\Delta x + \dots$$

(forward difference f' approx...)

- Also can use a different expansion to estimate f':

$$f(x_0 - \Delta x) = f(x_0) - f'(x_0)\Delta x + \frac{f''(x_0)}{2!}\Delta x^2 + \dots$$

$$\frac{f(x_0) - f(x_0 - \Delta x)}{\Delta x} = f'(x_0) + \frac{f''}{2!}\Delta x + \dots$$

(backward difference)

- The Taylor Series approach is generally applicable to produce difference formulas for derivatives of any order at any accuracy level.
- For example we may develop an expression for the third derivative:

$$f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{1}{2!}f''(x)\Delta x^2 + \frac{1}{3!}f'''\Delta x^3 + \frac{1}{4!}f''''\Delta x^4 + \dots$$

- For a third derivative estimate we need to remove f', f'' from these equations. (it will have $\mathcal{O}(\Delta x)$ accuracy)

(1) $$f_{i+2} = f_i + f' 2\Delta x + \frac{f''}{2!} 4\Delta x^2 + \frac{f'''}{3!} 8\Delta x^3 + \mathcal{O}(\Delta x^4)$$

(2) $$f_{i+1} = f_i + f' \Delta x + \frac{f''}{2!} \Delta x^2 + \frac{f''' \Delta x^3}{3!} + \mathcal{O}(\Delta x^4)$$

(3) $$f_{i-1} = f_i - f' \Delta x + \frac{f''}{2!} \Delta x^2 - \frac{f''' \Delta x^3}{3!} + \mathcal{O}(\Delta x^4)$$

- We aim to eliminate f', f'' from these to solve for f''' (so we have 3 eqns. & 3 unknowns). $f_{i+2}, f_{i+1}, f_i, f_{i-1}$ are assumed to be given.

- Alternatively we could eliminate $\mathcal{O}(\Delta x^4)$ by adding another eqn.

$$\mathcal{O}(\Delta x^4) = \frac{f^{(4)}}{4!} \Delta x^4 + \frac{f^{(5)}}{5!} \Delta x^5 + \dots$$

(in eqn for f_{i+1})

(4) $$f_{i-2} = f_i - 2f' \Delta x + \frac{f''}{2!} 4\Delta x^2 - \frac{f'''}{3!} 8\Delta x^3 + \frac{f^{(4)}}{4!} 16\Delta x^4 + \mathcal{O}(\Delta x^5)$$

- Now you have 4 eqns. & 4 unknowns.
  ↳ $(f, f', f'', f^{(4)})$

- Finite difference formulas can be applied iteratively to derive higher-order differences. E.g.:

$$f' \approx \frac{f_{i+1} - f_{i-1}}{2\Delta x} + \mathcal{O}(\Delta x^2)$$

$$f'' \approx \frac{(f')_{i+1/2} - (f')_{i-1/2}}{\Delta x}$$

→ { some approximation of derivative @ half point (2nd order accurate)

$$f'_{i+1/2} = \frac{f_{i+1} - f_i}{\Delta x} \quad \text{(centered formula)}$$

$$f'' \approx \frac{\dfrac{f_{i+1} - f_i}{\Delta x} - \dfrac{f_i - f_{i-1}}{\Delta x}}{\Delta x}$$

- This approach allows differentiation of complicated expressions like:

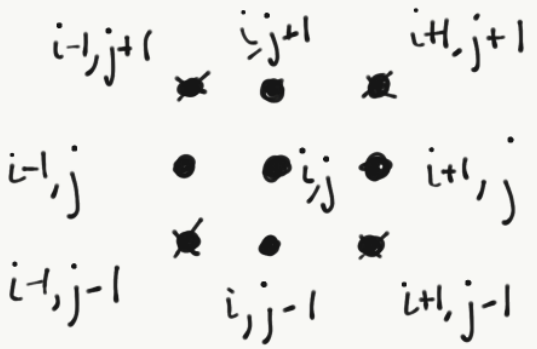
$$\frac{d}{dx}\left(A(x)\frac{df}{dx}\right) \quad \& \text{ similar.}$$

- Iterative applications of difference formula are also useful for deriving mixed derivatives

$$\left[\frac{\partial^2 f}{\partial x \partial y}\right]_{i,j} \approx \frac{1}{2\Delta x}\left(\left[\frac{\partial f}{\partial y}\right]_{i+1,j} - \left[\frac{\partial f}{\partial y}\right]_{i-1,j}\right)$$

$$\left[\frac{\partial f}{\partial y}\right]_{i+1} \approx \frac{(f_{i+1,j+1} - f_{i+1,j-1})}{2\Delta y}$$

$$\therefore \quad \left[\frac{\partial^2 f}{\partial x\,\partial y}\right]_{i,j} = \frac{1}{4\Delta x \Delta y}\left(f_{i+1,j+1} - f_{i+1,j-1} - f_{i-1,j+1} + f_{i-1,j-1}\right)$$



other approximations are possible this one preserves:

$$\left[\frac{\partial^2 f}{\partial x\,\partial y}\right]_{i,j} = \left[\frac{\partial^2 f}{\partial y\,\partial x}\right]_{i,j}$$

- Another approach would be to just differentiate iteratively: